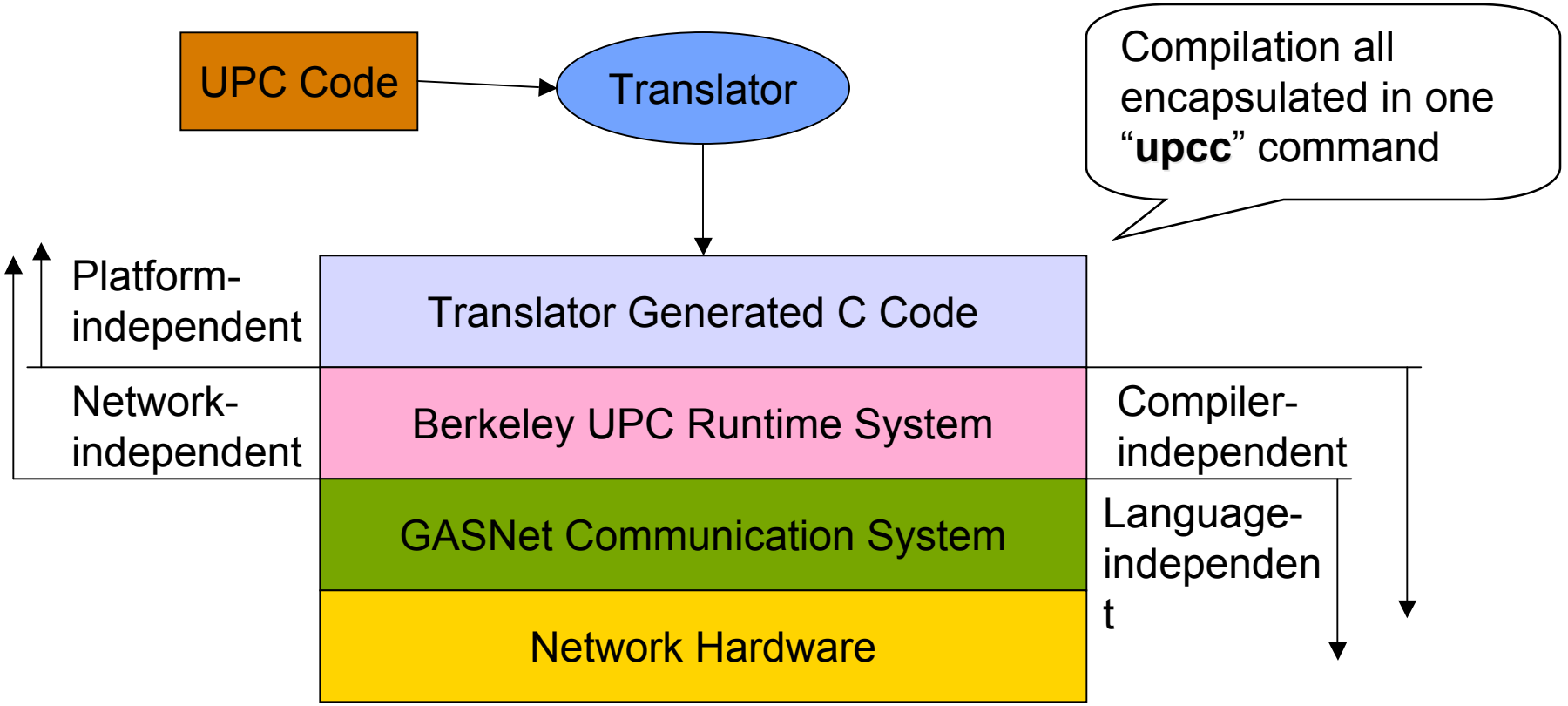
The Berkeley UPC Compiler: Implementation and Performance

Wei Chen
the LBNL/Berkeley UPC Group

Overview of Berkeley UPC Compiler

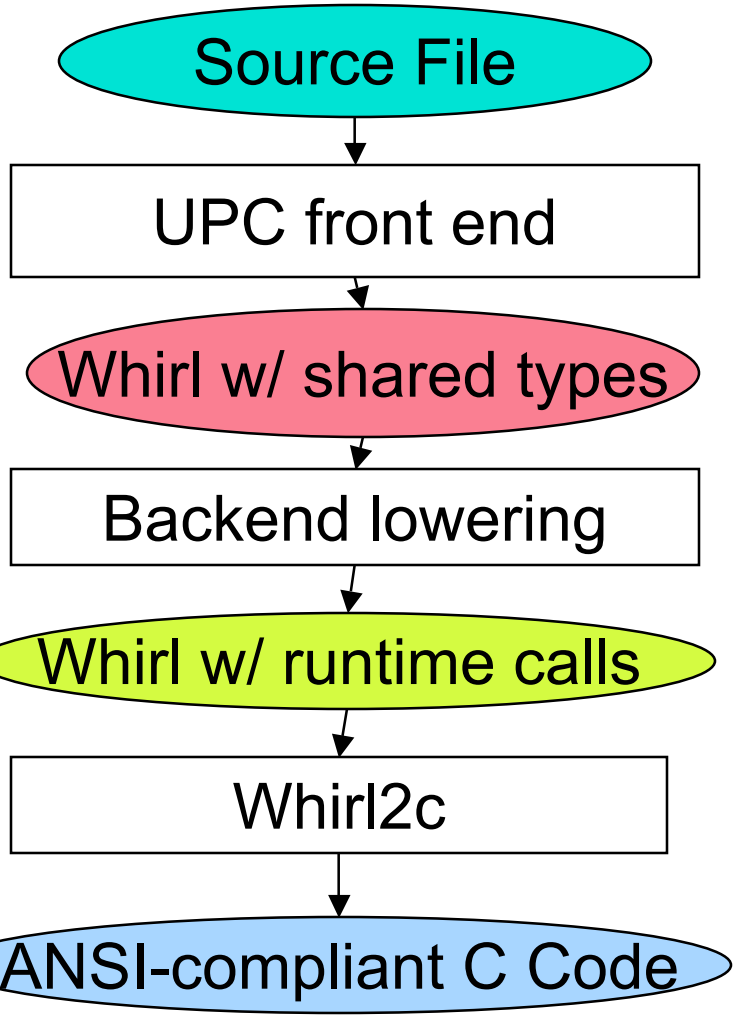


Two Goals: Portability and High-Performance

A Layered Design

- UPC to C translator: Translates UPC code into C, inserting calls to the runtime library for parallel features
- UPC runtime: Allocate/initialize shared data, perform operations on pointer-to-shared
- GASNet: An uniform interface for low-level communication primitives
- Portable:
  - C is our intermediate language
  - GASNet itself has a layered design with a small core
- High-Performance:
  - Native C compiler optimizes serial code
  - Translator can perform communication optimizations
  - GASNet can access network directly

Implementing the UPC to C Translator



- Based on Open64
- Supports both 32/64 bit platforms
- Designed to incorporate existing optimization framework in open64 (LNO, IPA, WOPT)
- Communicate with runtime via a standard API and configuration files
- Will present our implementation in Open64 workshop in March

Components in the Translator

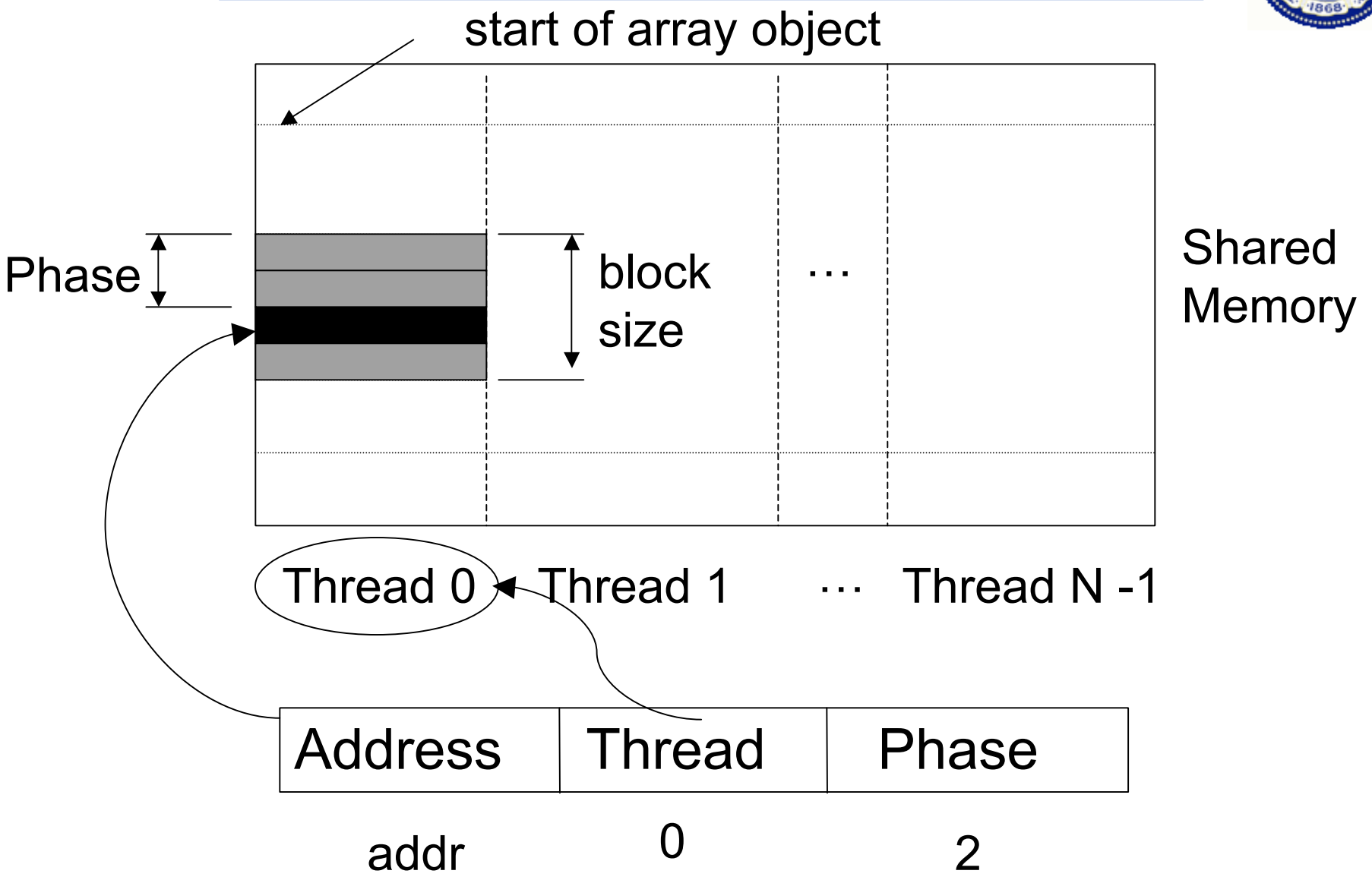
- **Front end:**
  - UPC extensions to C:
    shared qualifier, block size, forall loops, builtin functions and values (THREADS, memget, etc), strict/relaxed
  - Parses and type-checks UPC code, generates Whirl, with UPC-specific information available in symbol table
- **Backend:**
  - Transform shared read and writes into calls into runtime library. Calls can be blocking/non-blocking/bulk/register-based
  - Apply standard optimizations and analyses
- **Whirl2c:**
  - Convert Whirl back to C, with shared variables declared as opaque pointer-to-shared types
  - Special handling for static user data

Pointer-to-Shared: Phases

- UPC has three different kinds of distributed arrays:
  - Block-cyclic:
    shared [4] double a [n];
  - Cyclic:
    shared double a [n];
  - Indefinite (local to allocating thread):
    shared [] double *a = (shared [] double *) upc_alloc(n);
- A pointer needs a “phase” to keep track of where it is in a block
  - Source of overhead for updating and dereferencing
- Special case for “phaseless” pointers
  - Cyclic pointers always have phase 0
  - Indefinite blocked pointers only have one block
  - Don’t need to keep phase in pointer operations for cyclic and indefinite
  - Don’t need to update thread id for indefinite

phaseless

Accessing Shared Memory in UPC



Pointer-to-Shared Representation

- Important to performance, since it affects all shared operations
- Shared pointer representation trade-offs
  - Use of scalar types rather than a struct may improve backend code quality
    - Faster pointer manipulation, e.g., ptr+int as well as dereferencing
    - These are important in C, because array reference are based on pointers
  - Smaller pointer size may help performance
    - Use of packed 8-byte format may allow pointers to reside in a single register
    - But very large machines may require a longer representation

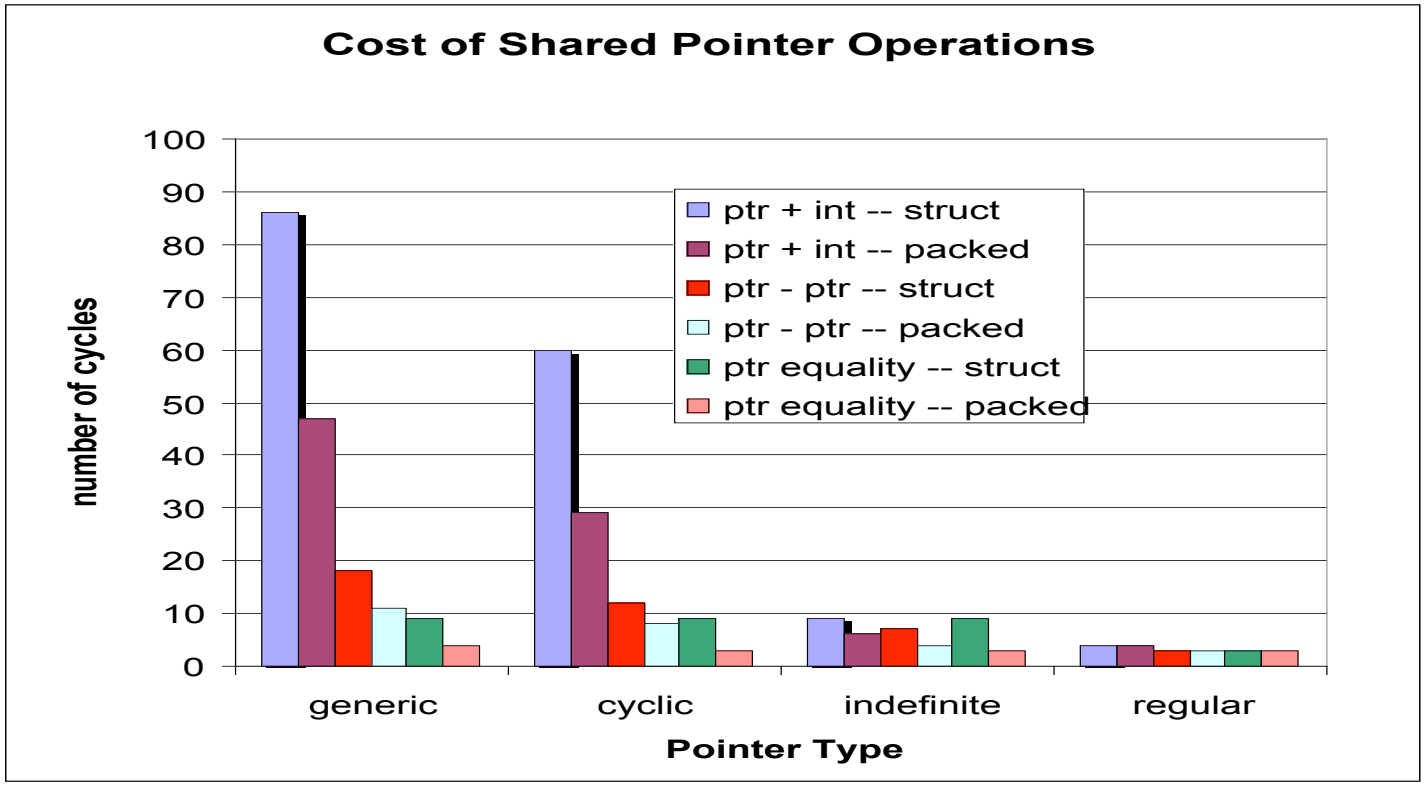
Let the Users Decide

- Compiler offers two pointer-to-shared configurations
  - Packed 8-byte format that gives better performance
  - Struct format for large-scale programs
- Portability and performance balance in UPC compiler
  - Representation is hidden in the runtime layer
  - Can easily switch at compiler installation time
  - Modular design means easy to add new representations (packed format done in one day)
  - May have a different representation for phaseless pointers (skipping the phase field)

Preliminary Performance

- Testbed
  - Compaq AlphaServer in ORNL, with Quadrics conduit
  - Compaq C compiler for the translated C code
- Microbenchmarks
  - Measures the cost of UPC language features and constructs
  - Shared pointer arithmetic, forall, allocation, etc
  - Vector addition: no remote communication
- Performance-tuning benchmarks (Costin)
  - Measure the effectiveness of various communication optimizations
  - Scale: test message pipelining and software pipelining
- NAS Parallel Benchmarks (Parry)
  - EP: no communication
  - IS: large bulk memory operations
  - MG: bulk memput

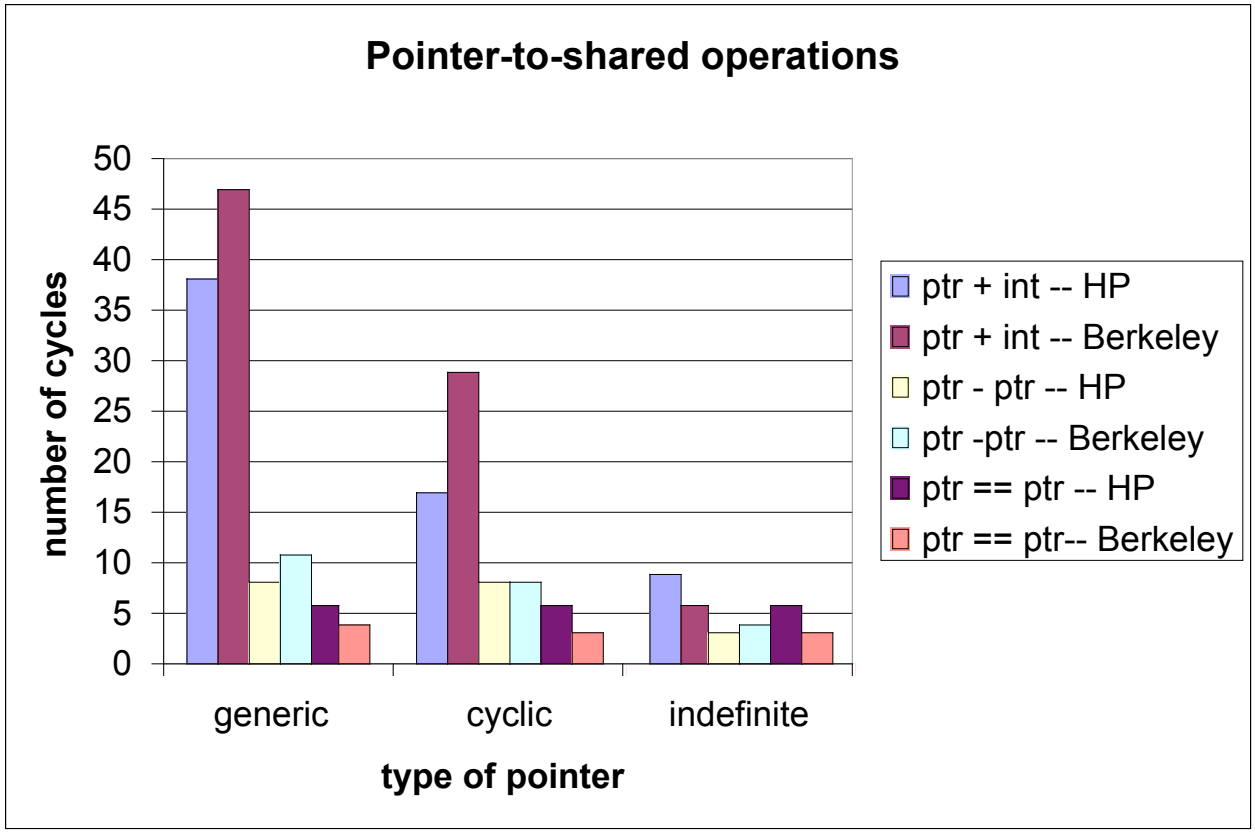
Performance of Shared Pointer Arithmetic



1 cycle = 1.5ns

- Phaseless pointer an important optimization
  - Indefinite pointers almost as fast as regular C pointers
- Packing also helps, especially for pointer and int addition

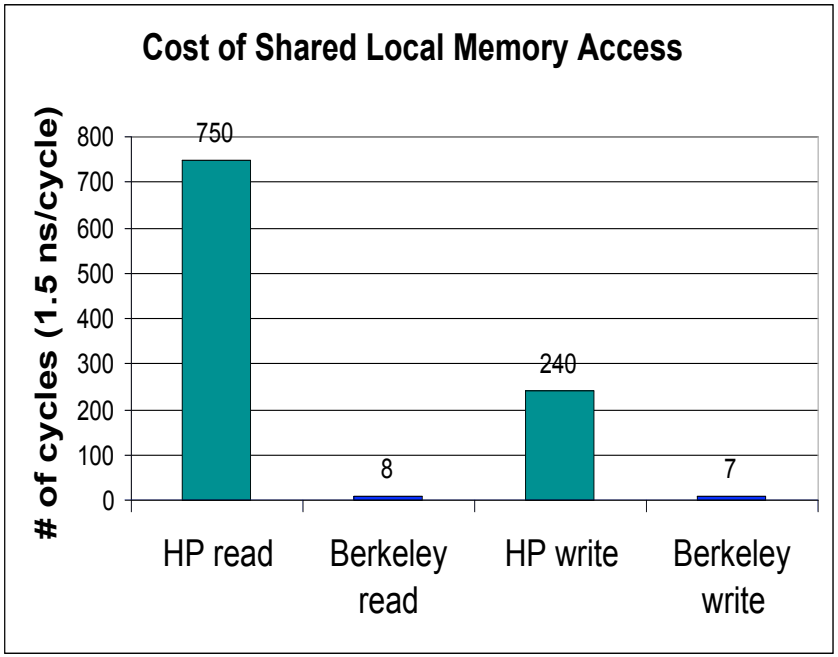
Comparison with HP UPC v1.7

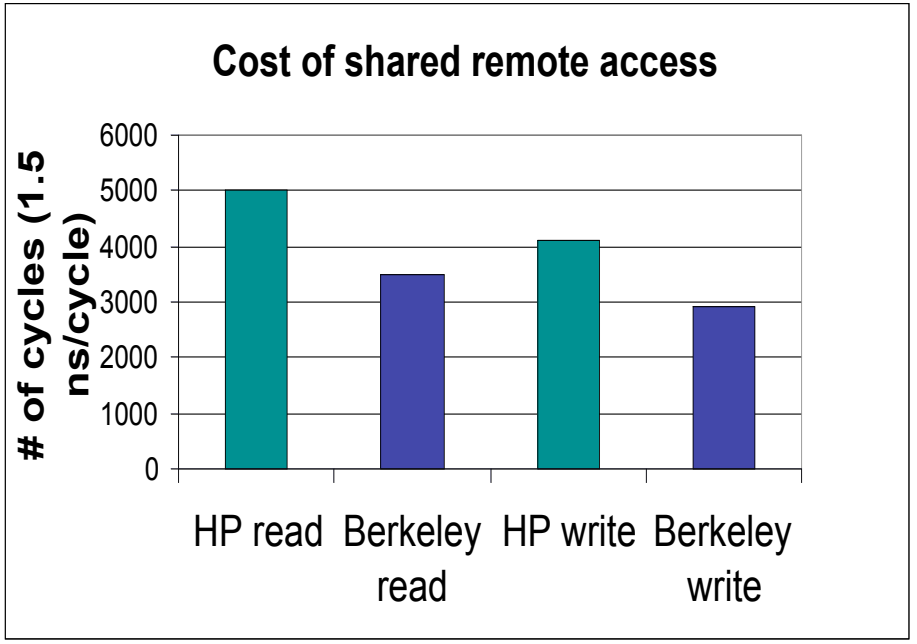


1 cycle = 1.5ns

- HP a little faster, due to it generating native code
- Gap for addition likely smaller with further hand-tuning

Cost of Shared Memory Access





- Local accesses somewhat slower than private accesses
  - HP has improved local access performance in new version
- Remote accesses worse than local, as expected
  - Runtime/GASNet layering for portability is not a problem

UPC Loops

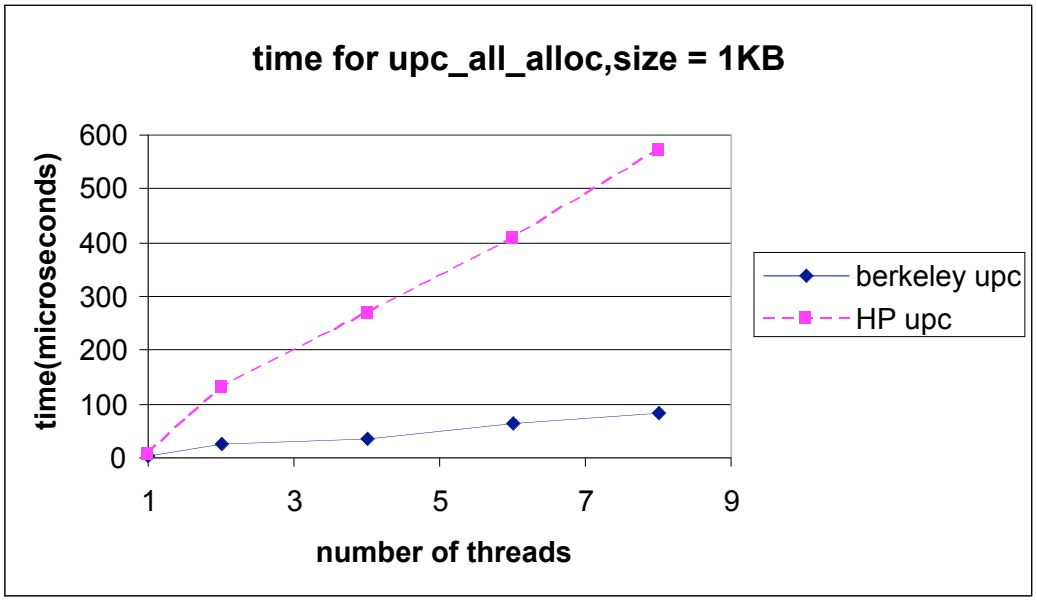
- UPC has a "forall" construct for distributing computation

```
shared int v1[N], v2[N], v3[N];
upc_forall(i=0; i < N; i++; &v3[i])
    v3[i] = v2[i] + v1[i];
```

- Two kinds of affinity expressions:
  - Integer (compare with thread id)
  - Shared address (check the affinity of address)
- Affinity tests are performed on every iteration

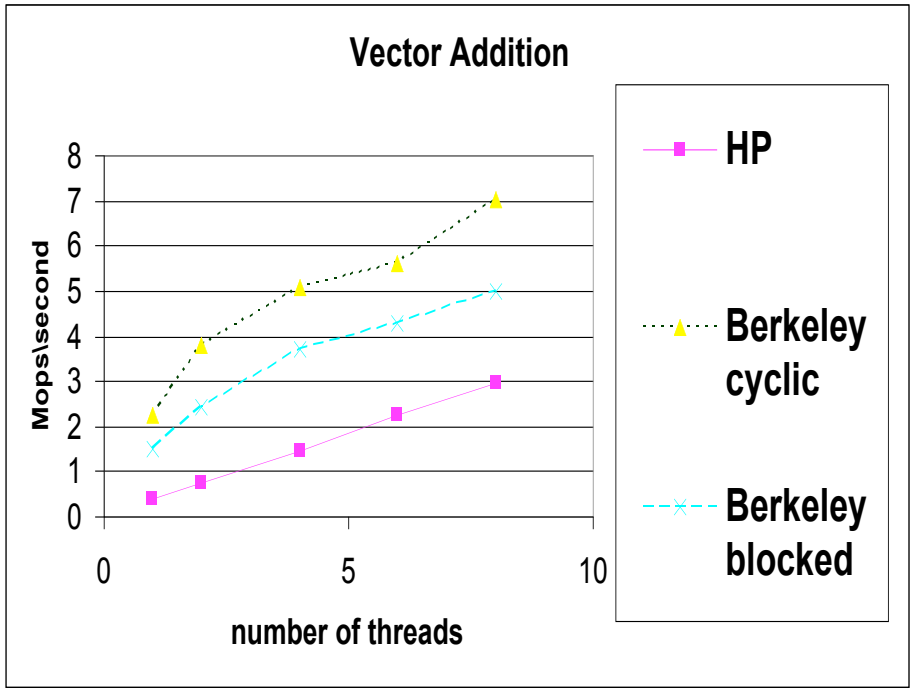
Affinity Exp	None	integer	shared address
# cycles	6	17	10

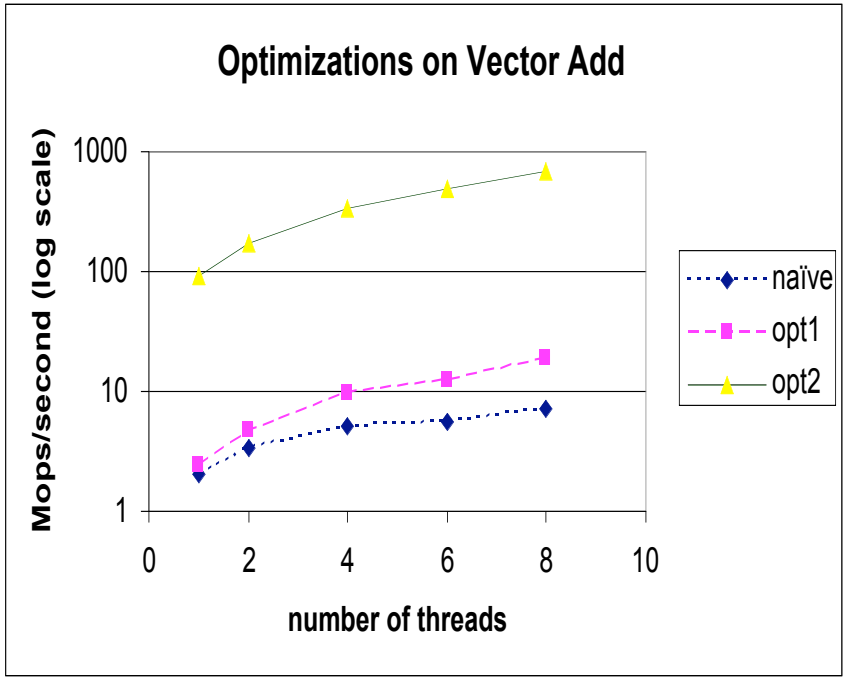
Overhead of Dynamic Allocation



- Faster than HP, due to the benefit of Active Messages
  - Shared allocation functions easy to implement, and scale well
  - Should perform better once the collectives (broadcast)are added to GASNet
- Shared locks also easy to implement using AM

Overhead of Local Shared Accesses





- Berkeley beats HP, but neither performs well
- Culprit: cost of affinity test and pointer-to-shared
- Privatizing local shared accesses improves performance by an order of magnitude

Observations on the Results

- Acceptable overhead for shared memory operations and access latencies
- Phaseless pointers are good for performance
- Packed representation is also effective
- Good performance compared to HP UPC 1.7
- Still lots of opportunities for optimizations

Compiler Status

- Targeting a 3/31 release that is fully UPC 1.1 compliant
- Compiler builds with gcc 2.96 and 3.2 on Linux
  - remote compilation option for other platforms
- Runtime and GASNet tested on AlphaServer (Quadrics), Linux (Myrinet), and IBM SP (LAPI)
- Successfully built and run NAS UPC benchmarks (EP, IS, CG, MG) ~ 2000 lines of code
- A paper submitted for publication

Challenges That We Solved

- **Portability is non-trivial to achieve**
  - **Double include: translator can't simply output declarations in system headers, because the runtime/GASNet may #include it**
  - **Porting the translator: Open64 originally only compiled for gcc2.96**
  - **IA-32 Support: Open64 was designed to generate IA-64 code**
  - **Preprocessor issues: Open64's C front end was not ANSI-compliant**
  - **Static User Data: Elaborate scheme to allocate and initialize static global data**
  - **Memory Management, Machine-specific information, and many more.**

Future Work: Optimizations

- Overlapping Communication with Computation
  - Separate get(), put() as far as possible from sync()
- Privatizing Accesses for Local Memory
  - Can be done in conjunction with elimination of forall loop affinity tests
- Message Pipelining
  - Effectiveness varies based on the network
- Message Coalescing/Aggregation/Vectorization
  - Reduce the number of small message traffic
- Prefetching and Software Caching
  - Difficulty is in understanding the UPC memory model

Future Work: Functionality

- Pthread and System V Shared Memory Support
- Port the translator to more platforms
- Debugging Support
- Merge with ORC (Open Research Compiler) to get new optimizations and bug fixes
- (Possible) Native Code Generation for IA64